Regular Expressions for Information Processing in ABAP

Ralph Benzinger
SAP AG

Regular Expression Primer

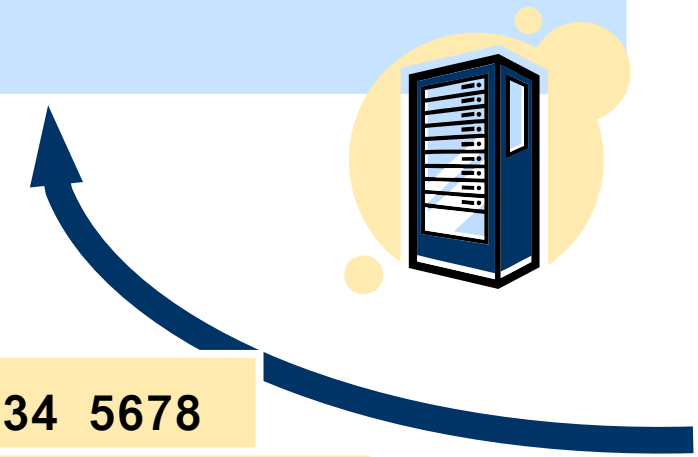
Using Regular Expressions in ABAP

Working with Regular Expressions

Ubiquitous Text Processing (1)

Checking input for validity:
Does credit card number contain only digits, hyphens, and spaces?

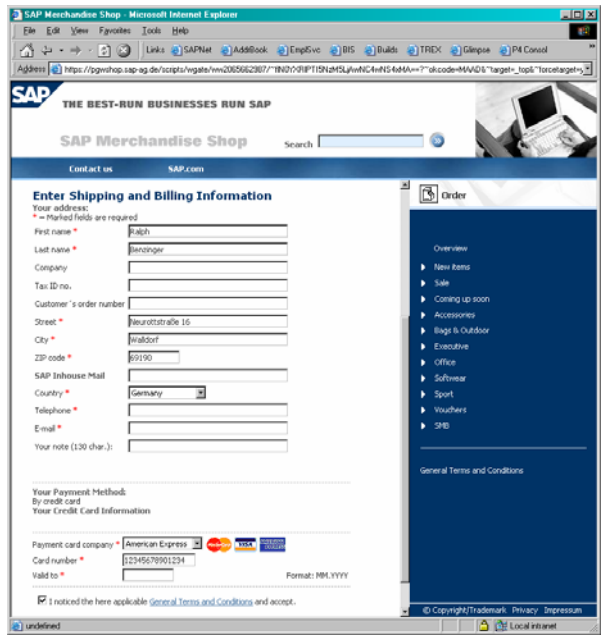
```
METHODS check
  IMPORTING cardno TYPE c
  RETURNING result TYPE abap_bool.
"…
```



1234 5678 1234 5678

1234 5678 1234 XXXX

1234-5678-1234-5678



Ubiquitous Text Processing (2)

Extracting information from text:
What is the document ID requested by the web client?

```
METHODS retrieve
  IMPORTING url TYPE c
  RETURNING doc TYPE xstring.

"…
```


```
http://sap.com/&user=ralph&id=1234&lang=EN
```

Ubiquitous Text Processing (3)

Normalizing values:
Eliminate non-digits in phone number for data export

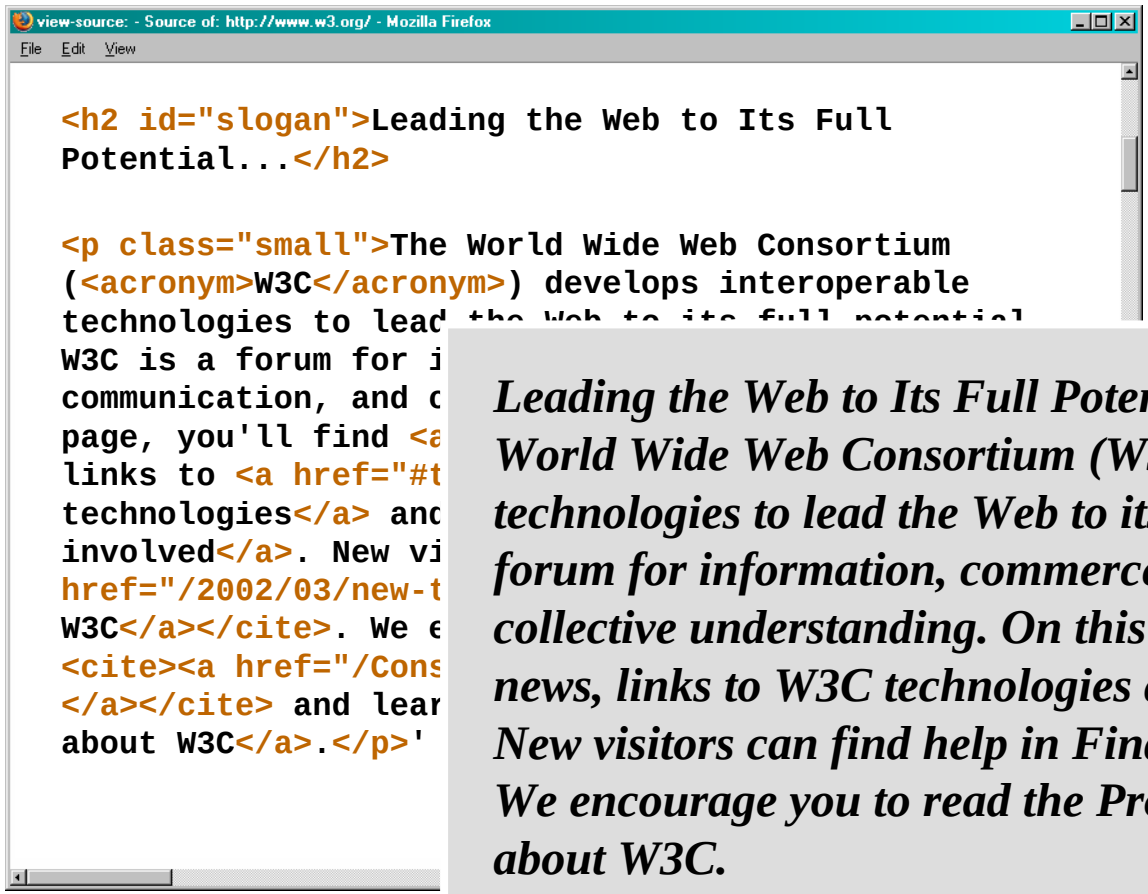
```
METHODS export
  IMPORTING phoneno TYPE c
  RETURNING digits  TYPE n.
"…
```

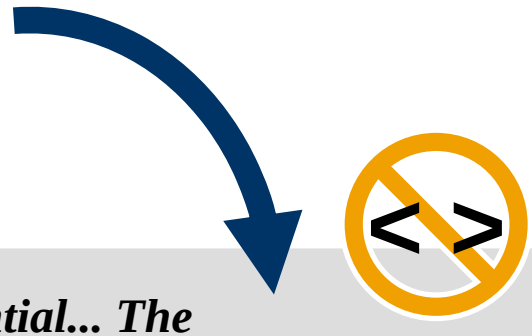
+49 (6227) 7-47474

496227747474

Sample Text Processing Problem

Remove all HTML tags from a given block of text





Leading the Web to Its Full Potential... The World Wide Web Consortium (W3C) develops interoperable technologies to lead the Web to its full potential. W3C is a forum for information, commerce, communication, and collective understanding. On this page, you'll find W3C news, links to W3C technologies and ways to get involved. New visitors can find help in Finding Your Way at W3C. We encourage you to read the Prospectus and learn more about W3C.

“Patterns? Been there, done that!”

Traditional ABAP offers some trivial patterns for text processing

*	one or more characters
+	any single character

- For use with CP operator
- Also used by F4 help

Sorry, but CP less useful than it seems

- Cx abbreviates “contains x”, but ...
- CP abbreviates “covers pattern” – does not search inside of text

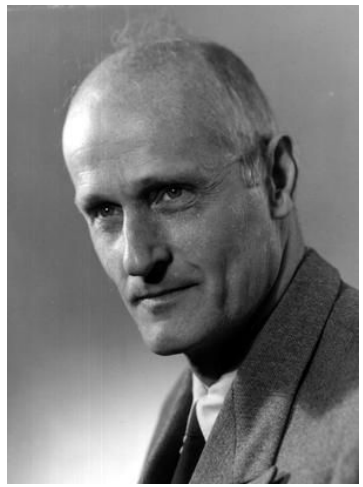
```
WHILE text CP '<*>'.
   " loop hardly ever entered
ENDWHILE.
```

- Pattern “ *<*>* ” won't tell us position of closing bracket

Improving Information Processing with REs

Regular Expressions (REs, regexes) provide powerful pattern language for text processing

- **Well understood**
  - Developed by mathematician Kleene in the 1950s
- **Powerful**
  - Highly focused special purpose language
  - Many common text processing tasks turn into simple one-liners
- **Standardized and widely used**
  - Made popular by Unix tools: *grep, sed, awk, emacs*
  - Built into some languages like *Perl* and *Java*; add-on libraries available for many others
  - Early Unix de-facto standard formalized in PCRE and POSIX



Stephen C. Kleene
Image © U. of Wisconsin

As of Release 7.00, ABAP supports POSIX-style regular expressions

Regular Expression Basics

Regular expressions are patterns built from

- Literals (characters)
- Operators (meta characters)

```
. * + ? | ^ $ ( ) [ ] { }  \
```

- Prepending " \ " turns operators into literals

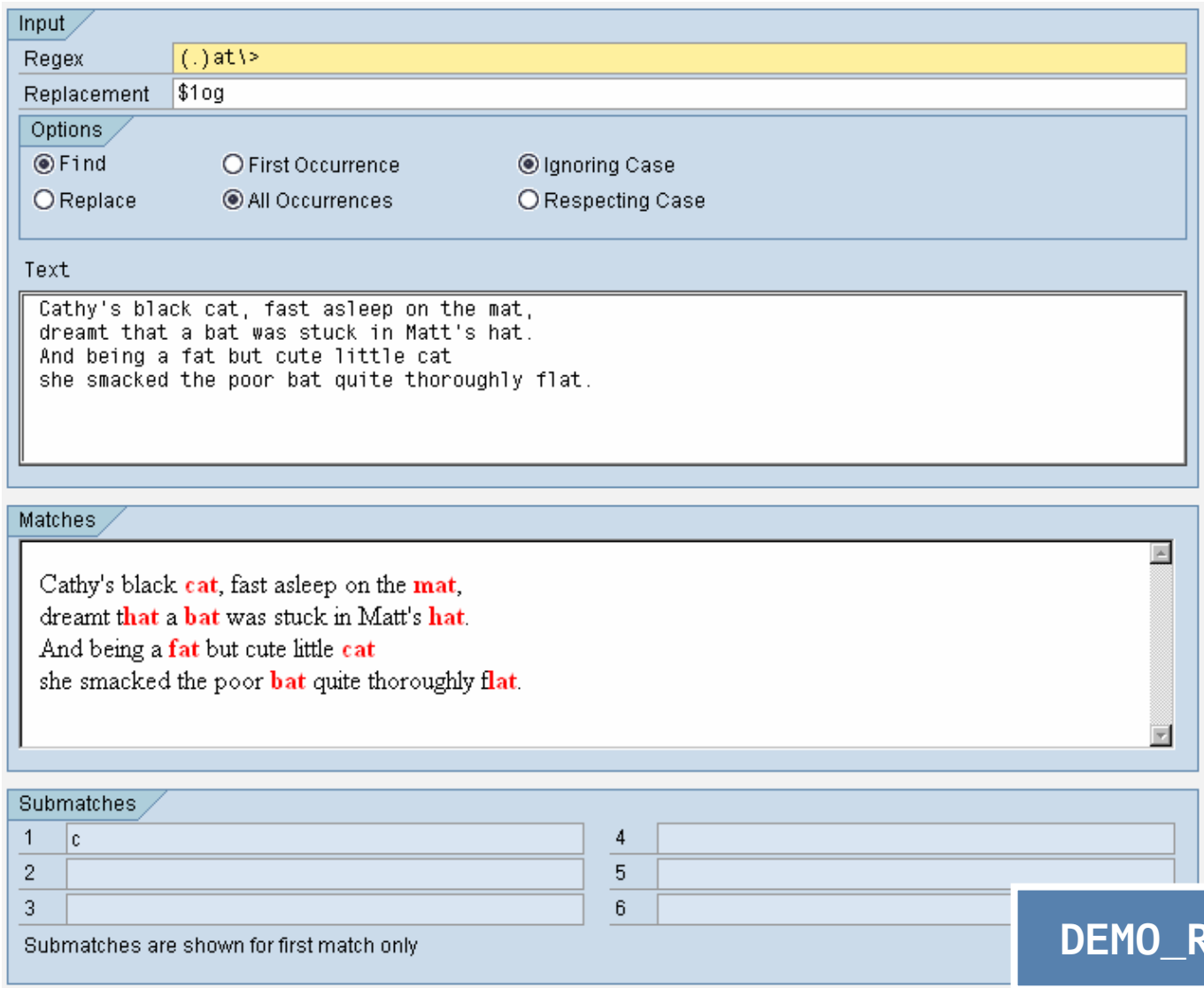
Basic regex terminology

- An RE represents a set of literal text strings
- RE matches text if complete text is represented by RE
- REs are commonly used for searching text
- Text to be searched may contain one or more matches

Covers

Contains

Tool Support: Regex Toy



DEMO_REGEX_TOY

Validating data with regular expressions

- Data dat is valid if and only if it matches against regex pat

```
IF cl_abap_matcher=>matches( pattern = pat
                             text    = dat )
   = abap_true.
   " accept valid input
ELSE.
   " reject invalid input
ENDIF.
```

- False positives = data is invalid but matches
- False negatives = data is valid but does not match

- Strike the right trade-off:

  Complexity of regex ← vs → Cost of false positives/negatives

Regular Expression Primer

Using Regular Expressions in ABAP

Working with Regular Expressions

Processing Text in ABAP

ABAP offers a variety of statements and operators for text processing

Statement	Semantics
FIND … IN …	find substring(s) in text
REPLACE … IN … WITH …	replace one or more substring(s) in text
REPLACE SECTION …	replace given section of text
CONCATENATE … INTO …	concatenate strings and fields
SPLIT … AT … INTO …	split text at given character
TRANSLATE …	convert case and substitute characters
CONDENSE …	remove extraneous whitespace
SHIFT …	move characters left or right
OVERLAY … WITH …	replace characters based on word mask
CS, CO, CA, CN, CP, …	contains…/covers… operators

Native regex support for FIND and REPLACE statements

- Finding first occurrence

```
FIND REGEX pattern IN text
     MATCH OFFSET off MATCH LENGTH len.
```

- Replacing all occurrences

```
REPLACE ALL OCCURRENCES OF REGEX pattern
        IN text WITH new
        REPLACEMENT COUNT cnt.
```

- Supports all known additions such as IGNORING CASE
- Additional support for FIND ALL OCCURRENCES
- Additional support for searching internal tables
- REs limited to CHARACTER MODE

Searching returns leftmost-longest match within text

```
FIND REGEX '.at' IN text.
```

```
The cat with the hat sat on Cathy's mat.
```

```
    cat
                hat
                    sat
                        Cat
                            mat
```

"Leftmost" takes precedence over "longest"

```
FIND REGEX '.at(\w|\s)*th' IN text.
```

```
The cat with Cathy's hat thus sat on the mat.
    cat with
    cat with Cath
                    hat th
                    hat thus sat on th
                            sat on th
```

Getting Information from FIND and REPLACE

Obtain individual information by using additions

```
FIND REGEX pattern
  IN [TABLE] text
  MATCH COUNT  cnt
  MATCH LINE   lin
  MATCH OFFSET off
  MATCH LENGTH len.
```

```
REPLACE REGEX pattern
  IN [TABLE] text WITH new
  REPLACEMENT COUNT  cnt
  REPLACEMENT LINE   lin
  REPLACEMENT OFFSET off
  REPLACEMENT LENGTH len.
```

- Contains information about the last match/replacement, if any
- Success indicated by sy-subrc and MATCH COUNT
- Get match text by offset and length access

```
text+off(len)
```

- Replacement information is about replacement text, not text replaced
- Not suitable for obtaining information on all matches/replacements

Searching for All Matches

Use " RESULTS itab " addition when searching for all matches

```
DATA res TYPE match_result_tab.
FIND ALL OCCURRENCES OF REGEX r IN text RESULTS res.
```

- Predefined DDIC data types match_result and match_result_tab

match_result			
line	offset	length	submatches
TYPE i	TYPE i	TYPE i	TYPE TABLE…

- Process result with LOOP AT

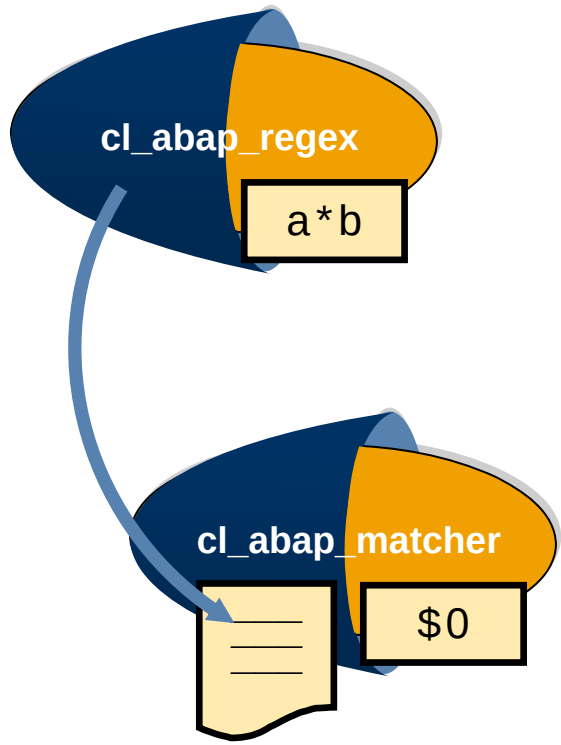
```
FIELD-SYMBOLS <match> TYPE match_result.

LOOP AT res ASSIGNING <match>.
  WRITE / text+<match>-offset(<match>-length).
ENDLOOP.
```

Using ABAP Regex Classes

ABAP Objects provides two classes for using REs in object-oriented programs

- Regex class cl_abap_regex
  - Stores preprocessed RE pattern for increased performance
  - Should be reused to avoid costly re-processing
- Matcher class cl_abap_matcher
  - Central class for interaction with REs
  - Links text to regex object
  - Stores copy of text to process (efficient for strings, costly for fixed-length fields)
  - Tracks matching and replacing within text



Using regex classes in ABAP

- Create regex and matcher and interact with matcher
- Use static class methods of matcher as a shorthand

Creating Regex Objects

There are two ways of setting up objects for RE processing

```
DATA: regex   TYPE REF TO cl_abap_regex,
      matcher TYPE REF TO cl_abap_matcher.
```

- **Using CREATE**

```
CREATE OBJECT regex   EXPORTING pattern     = 'a*b'
                                ignore_case = abap_true.
CREATE OBJECT matcher EXPORTING regex       = regex
                                text        = text.
```

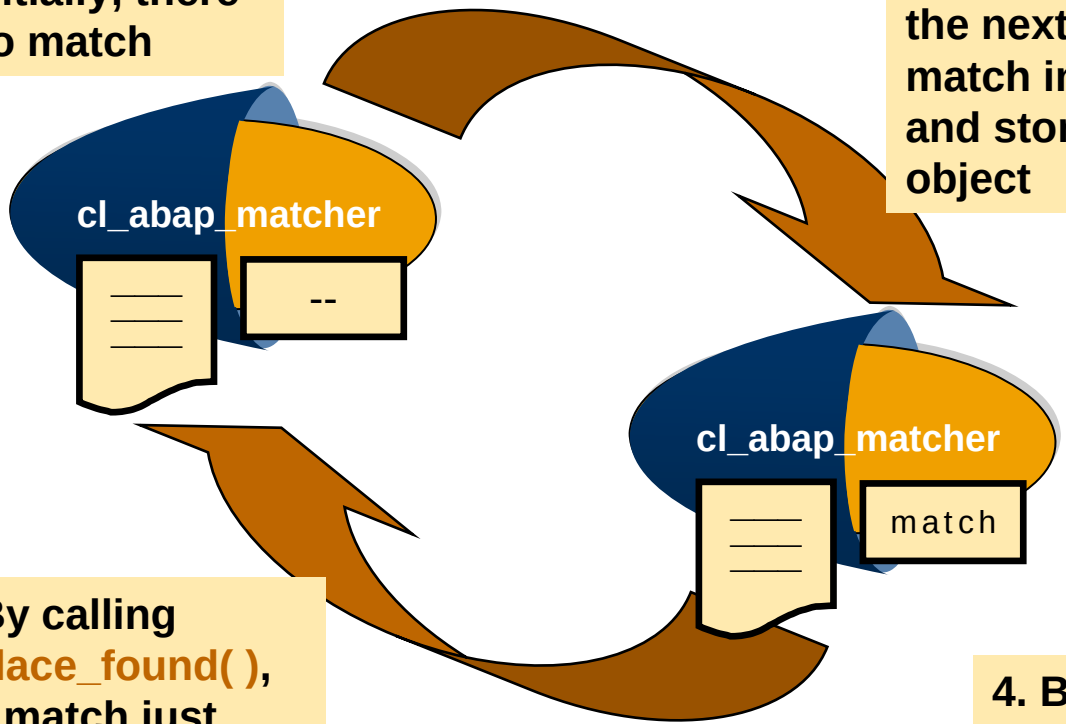
- **Using factory method**

```
matcher = cl_abap_matcher=>create( pattern = 'a*b'
                                   text    = text ).
```

Using Regex Objects for Matching

The matcher objects keeps track of the current match

1. Initially, there is no match



2. By calling find_next( ), the next unprocessed match in text is located and stored in the matcher object

3. Information about the current match can be retrieved by calling get_match( ).

4. By calling find_next( ) again, the matcher advances to the next unprocessed match

5. By calling replace_found( ), the match just found is replaced as specified

cl_abap_matcher interface

Finding	
find_next( )	find_all( )
match( )	
Replacing	
replace_found(new)	replace_all(new)
replace_next(new)	
Querying	
get_offset([index])	get_match( )
get_length([index])	get_submatch(index)
get_line( )	

Using Regex Class Methods

The matcher provides two class methods for logical expressions

```
cl_abap_matcher=>contains( pattern text [options] )
cl_abap_matcher=>matches(  pattern text [options] )
```

- Class methods return type `abap_bool`
- If successful, match information can be obtained with `get_object( )`

```
DATA: matcher TYPE REF TO cl_abap_matcher,
      match   TYPE match_result.
IF cl_abap_matcher=>contains( pattern = 'a*b'
                              text = text ) = abap_true.
    matcher = cl_abap_matcher=>get_object( ).
    match   = matcher->get_match( ).
    WRITE / matcher->text+match-offset(match-length).
ENDIF.
```

Regular Expression Primer

Using Regular Expressions in ABAP

Working with Regular Expressions

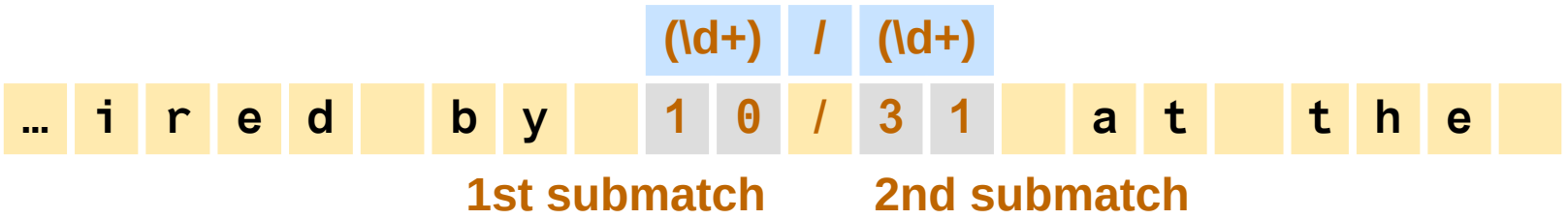
Submatches

Parentheses capture submatches for later reference

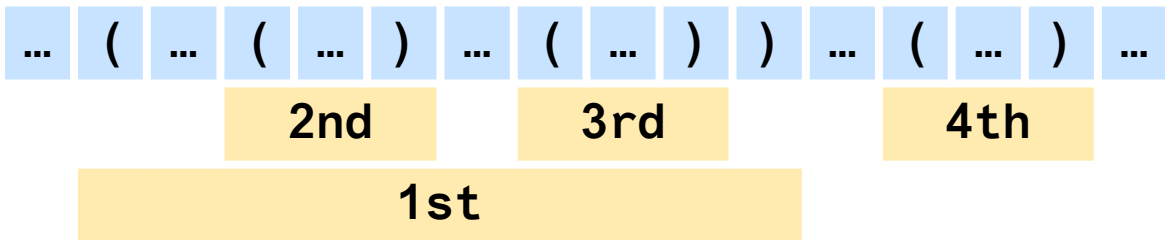
- One of the most useful features of REs

`(\d+)/(\d+)`	… required by 10/31 at the latest …

- Match is subdivided into submatches and stored internally



- Groups numbered left-to-right by opening parenthesis



- No upper limit on number of groups

Querying Submatches

- **Using FIND with RESULTS addition**

```
DATA match TYPE match_result.
FIND REGEX '(\d+)/(\d+)' IN text RESULTS match.
```

match_result-submatches		
TYPE TABLE OF	submatch_result	
	offset	TYPE i
	length	TYPE i

- **Using matcher class**

	get_offset([n])	get_submatch(n)
n > 0	offset of n-th submatch	text of n-th submatch
n = 0	offset of complete match	text of complete match

ABAP removes some stumbling blocks from POSIX REs

- Unlimited number of subgroups (POSIX only supports \1 through \9)

```
REPLACE REGEX '…\123…' IN text WITH '…$456…'.
```

- No control code escaping: \n denotes n-th subgroup, not char code n
- No collating elements or equivalence classes

ABAP searches exhaustively for matches, Perl stops at first match

- Perl returns leftmost-first match

ABAP	b+|a+b|[ab]+	oaaabbbo
Perl	b+|a+b|[ab]+	oaaabbbo

- Perl-style matching is sometimes faster, but generally harder to use
- Perl-style matching can be simulated with cut operator " (?>) "

Limitations

How could there be anything that regular expressions cannot do?

Cannot count unlimited quantities

- **Does text contain as many a's as b's?**
- **Does text contain more a's than b's?**

Close, but no cigar:

```
((a+b)|(ba+))+
```

Cannot remember unlimited amount of previously matched text

- **Are parentheses well-balanced? (((())(()))())**
- **Is text a palindrome (reading the same forward and backward)?**
  - **With back references we can match fixed-length palindromes, though**

Recommendation: Build missing functionality in ABAP!

Going Overboard

So you'd like to reject all invalid dates

```
2/30/2006
```

Would you use this regex then?

```
^(?=\d)(?:(?:31(?!.(?:0?[2469]|11))|(?:30
|29)(?!.0?2)|29(?=.0?2.(?:(?:(?:1[6-9]|[2
-9]\d)?(?:0[48]|[2468][048]|[13579][26])|
(?:(?:16|[2468][048]|[3579][26])00)))(?:\
x20|$))|(?:2[0-8]|1\d|0?[1-9]))([-./])(?:
1[012]|0?[1-9])\1(?:1[6-9]|[2-9]\d)?\d\d(
?:(?=\x20\d)\x20|$))?((0?[1-9]|1[012])(:
[0-5]\d){0,2}(\x20[AP]M))|([01]\d|2[0-3])
(:[0-5]\d){1,2})?$
```

Some things are best left to ABAP!

Common Regex Pitfalls

Most regex errors can be attributed to oversight or omission:

- **Matches are greedy**
- **Matches may be empty**
- **Matches may be found anywhere**
- **The value range may be larger than expected**
- **Part delimiters may be escaped or quoted**

Summary

- Regular expressions are powerful text processing tool
  - Validation
  - Extraction
  - Transformation
- Writing effective REs involves
  - Using alternatives, quantifiers, sets, classes
  - Grouping parts into submatches
  - Restricting matches with anchors, classes, look-aheads
  - Avoiding common pitfalls
- ABAP supports POSIX-style regular expressions
  - FIND and REPLACE
  - cl_abap_regex and cl_abap_matcher
  - ABAP Workbench tools

Bibliography

Resources

- NetWeaver ABAP Online Documentation
- Two-article series on regular expressions to be published in SAP Professional Journal

recom-
mended

Books

- Jeffrey Friedl: Mastering Regular Expressions, O'Reilly
- John Hopcroft and Jeffrey Ullman: Introduction to Automata Theory, Languages, and Computation, Addison Wesley

Questions?

Q&A